Chapter 11
AI-Completeness: Using Deep Learning to Eliminate the Human Factor

Kristina Šekrst

Abstract Computational complexity is a discipline of computer science and mathematics which classifies computational problems depending on their inherent difficulty, i.e. categorizes algorithms according to their performance, and relates these classes to each other. **P** problems are a class of computational problems that can be *solved* in polynomial time using a deterministic Turing machine while solutions to **NP** problems can be *verified* in polynomial time, but we still do not know whether they can be solved in polynomial time as well. A solution for the so-called **NP**-complete problems will also be a solution for any other such problems. Its artificial-intelligence analogue is the class of **AI**-complete problems, for which a complete mathematical formalization still does not exist. In this chapter we will focus on analysing computational classes to better understand possible formalizations of **AI**-complete problems, and to see whether a universal algorithm, such as a Turing test, could exist for all **AI**-complete problems. In order to better observe how modern computer science tries to deal with computational complexity issues, we present several different deep-learning strategies involving optimization methods to see that the inability to exactly solve a problem from a higher order computational class does not mean there is not a satisfactory solution using state-of-the-art machine-learning techniques. Such methods are compared to philosophical issues and psychological research regarding human abilities of solving analogous **NP**-complete problems, to fortify the claim that we do not need to have an exact and correct way of solving **AI**-complete problems to nevertheless possibly achieve the notion of strong AI.

11.1 Learning How to Multiply

The notion of *computation* has existed in some form since the dawn of mankind, and in its usual meaning, the term itself refers to a way of producing an output from a set of inputs in a finite number of steps. Computation is not just a practical tool for everyday life but also a major scientific concept since computational experts

K. Šekrst (✉)
University of Zagreb, Zagreb, Croatia
e-mail: ksekrst@ffzg.hr

S. Skansi (ed.), *Guide to Deep Learning Basics*,
https://doi.org/10.1007/978-3-030-37591-1_11

realized that many natural phenomena can be interpreted as computational processes [2]. *Computational complexity theory* classifies computational problems in line with their inherent difficulty. In computational complexity theory, a decision problem is a problem that gives a *yes/no* answer for the input values, for example, given a number x, decide if x is a prime number. Decision problems that can be solved by an algorithm are *decidable*. Some ways of solving a problem are better, i.e. more efficient, than others.

Let us start with a simple problem of basic multiplication: given two integers, compute their product. We can just repeat addition, for example, $5 \times 4 = 4 + 4 + 4 + 4 + 4$. But things are getting complicated with examples such as 4575852×15364677. Usually people think that the grade-school method is the only one, but this is far from the truth. Historically, computers have used shift-and-add algorithms for multiplication issues, but their computing powers needed to get faster since the complexity of many computational problems amounts to the speed of multiplication. The grade-school method is carried out in n^2 steps, for n number of digits, which becomes an issue for everyday computations of millions of digits. In 1960, Andrey Kolmogorov conjectured that the standard multiplication procedure requires a number of elementary operations proportional to n^2, i.e. $O(n^2)$ in the big O notation,[1] which describes how the running time of algorithms grows as their input size grows. It is easy to calculate n^2 if $n = 2$, but it is not that fast if n has a billion digits. At the Moscow State University, Kolmogorov had organized a seminar on computational problems and introduced his famous conjecture, but within a week, Anatoly Karatuba, then a 23-year-old student, disproved it by finding an algorithm that multiplies two n-digit numbers in $O(n \log_2 3) \approx (n^{1.585})$ elementary steps [14].

Karatsuba's method uses a divide-and-conquer approach by dividing the problem into sub-problems, solving the sub-problems, and combining the answer to solve the original problem. First, we take numbers x and y, for example, 58×63, with their bases B:

$$\begin{aligned} x &= x_1 \times B + x_2 & \qquad y &= y_1 \times B + y_2 \\ x &= 58 & y &= 63 \\ x &= 5 \times 10 + 8 & y &= 6 \times 10 + 3 \end{aligned}$$

The product now becomes $x \times y = (x_1 \times B + x_2)(y_1 \times B + y_2)$, which we are splitting into smaller computational blocks:

$$\begin{aligned} a &= x_1 \times y_1 \\ b &= x_1 \times y_2 + x_2 \times y_1 \\ c &= x_2 \times y_2 \end{aligned}$$

[1] A typical usage of the big O notation is asymptotical and refers to the largest input value since its contribution grows the fastest and makes other inputs irrelevant.

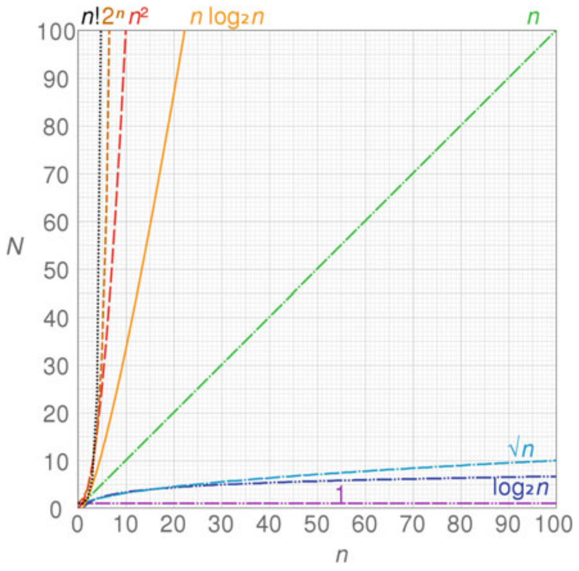

Fig. 11.1 We would like a graph of our computational-problem algorithm to run as low as possible such that there is a huge step for our resources from $O(n^2)$ to $O(n \log n)$, which means there is a significantly smaller number of operations for the input of size n. *Source* www.commons.wikimedia.org, CC BY-SA 4.0

Karatsuba discovered that b may be shortened to $b = (x_1 + x_2)(y_1 + y_2) - a - c$, which is a key step that now gives us two multiplications less, instead of $b = x_1 \times y_2 + x_2 \times y_1$!

$$
\begin{aligned}
a &= 5 \times 6 = 30 \\
b &= (5 + 8) \times (6 + 3) - 5 \times 6 - 3 \times 8 = 63 \\
c &= 3 \times 8 = 24 \\
x \times y &= a \times B^2 + b \times B + c \\
x \times y &= 30 \times 10^2 + 63 \times 10 + 24 = 3654
\end{aligned}
$$

Karatsuba's approach made way for even better methods, such as Schönhage and Strassen's method [20], whose runtime is $O(n \log n \log \log n)$ for n-digit numbers, which uses fast Fourier transforms. In 2019, Harvey and van der Hoeven [11] proved that you can achieve integer multiplication in $O(n \log n)$.[2] This example illustrates how even small modifications can be crucial to lower the computational complexity of an algorithm (Fig. 11.1).

Computational complexity theory deals with the resources required during computation to solve a computational problem, both temporal (how many steps we need to solve a problem) and spatial (how much memory we need to solve a problem). Problems of class **P** are those that can be *solved* using a deterministic Turing machine in a *polynomial* amount of time (for example, n^2, but not exponential 2^n). On the

[2] *Caveat*: it only performs faster than other algorithms for numbers with over 2^{4096} digits, i.e. bits, which is seldom practical even for big-data purposes.

other hand, **NP** problems have solutions that can just be *verified* in polynomial time but we still do not know whether they can also be *calculated* in polynomial time. **NP**-complete problems are the hardest **NP** problems, and an algorithm that can solve such a problem in polynomial time, can also *solve any other* **NP** problem in polynomial time. Usually, **NP**-complete problems require exponential time, for example, $O(2^n)$, which is easy when n is small but has rapid big jumps when n increases. For instance, consider a program that runs in 2^{10} hours, which amounts to 42.6 days. But if we increase n to 11, the result is 85 days, and if we increase n to 20, the program will finish in 119.6 years.

NP-hard problems are at least as hard as the hardest problems in **NP**. Usually, this amounts to **NP**-complete problems, but there are **NP**-hard problems which are not **NP**-complete, for example, *the halting problem* ("given a program and its input, will a program run forever?"), which is a decision problem that is *undecidable*.[3] The most famous **NP**-complete problems comprise, for example, *Boolean satisfiability problem* ("is there an interpretation that satisfies a given Boolean formula?"), *travelling-salesman problem* ("given a length L, decide whether the graph of cities and distances has any tour shorter than L?"), *knapsack problem* ("given a set of items with some weights and values, can a value of at least V be achieved without exceeding the weight W?"), and *graph-colouring problem* ("can we colour the graph vertices such that no two adjacent ones are of the same colour?"). A major unsolved problem in computer science is the ***P** versus **NP** problem*, which asks *whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time*, i.e. quickly.

11.2 AI-Complete

Analogous to **NP**-complete problems, the most difficult problems in the field of artificial intelligence are known as **AI**-complete, the term first coined by Fanya Montalvo [18]. Assuming intelligence is computational, to solve one of such problems would be equal to solving the central artificial intelligence problem, i.e. *strong AI*: the intelligence of a machine that has a human-like capacity to understand or learn any intellectual task. **AI**-complete problems usually include problems from computer vision or natural language understanding, along with automated reasoning, automated theorem proving, and dealing with unexpected circumstances while solving real-world problems. However, unlike the exact formalization of computational complexity classes in computer science, **AI**-complete problems have not been completely mathematically formalized.

[3]Presuppose we have a computable function (that solves the halting problem). That function runs a subroutine which detects whether our function will halt, and if that subroutine returns true, it should loop forever. If the function fulfils the condition of halting and returns true, then it will loop forever and never halt. However, if it returns false and does not halt, it will not loop forever, so it will immediately halt. These two contradictions then bring down the presupposition that it was a computable function.

Ahn et al. [1] presented a possible formalization by defining an AI problem as a triple $P = (S, D, f)$, where S is a set of problem instances, D is probability distribution over the problem set S, and $f : S \mapsto \{0, 1\}^*$ answers the instances. A function f maps problem instances to their set memberships, i.e. recognizing if the property in question has some given patterns. The authors give a caveat that defining an AI problem *should not be inspected with a philosophical eye* since they are not trying to capture all the problems that fall under the domain of AI. Nonetheless, they switch the focus to the AI community, that should agree on what hard AI problems really are. However, that does not mean that people have to be able to solve such tasks, but a crucial characteristic is that a certain fraction of a human population can solve it, without a temporal restriction.

Yampolskiy [27] has defined **AI**-complete problems using a Human Oracle (HO) function capable of computing any function computable by the union of all human minds, i.e. any cognitive ability of any human whatsoever can be repeatable by the HO. Hence, a problem **C** is **AI**-complete if it has two properties:

1. it is in the set of AI problems (Human-Oracle solvable) and
2. any AI problem can be converted into **C** by some polynomial-time algorithm.

A problem **H** is **AI**-hard if and only if there is an **AI**-complete problem **C** that is polynomial-time Turing reducible to **H**. And **AI**-easy problems are solvable in polynomial time by a deterministic Turing machine with an oracle for some AI problems. Yampolskiy shows that a Turing test problem is **AI**-complete since it is HO-solvable (which trivially follows from the definition of the test itself). For the second condition, it is needed to show that any problem solvable by the HO function could be encoded as an instance of the Turing test, which is a condition parallel to **AI**-complete problems, whose polynomial-time solutions could also be solutions to any **NP** problem. By taking the input as a question used in the Turing test, and output as an answer, any HO-solvable problem could be reduced in polynomial time to an instance of a Turing test. This kind of heuristics can be generalized in such a way that we can check whether all the information in questions that could be asked during a Turing test could be encoded as an instance of our current AI problem. That heuristics, for example, eliminates chess as an **AI**-complete problem since only limited information can be encoded in starting positions on a standard chessboard. Yampolskiy classifies question answering and speech understanding as **AI**-complete problems as well.

11.3 The Gap

Unlike in [1], we believe that defining an AI problem *should* be inspected with a philosophical eye and that it already has been inspected with a philosophical eye outside the formal context. We have stated that solving such problems would be equal to solving the strong-AI problem. Philosophy has been walking hand-to-hand with modern AI development since Norbert Wiener, a mathematician and a philosopher, who theorized that all intelligent behaviour, as a result of different feedback mechanisms, could be simulated by a machine. However, philosophers were also debating that consciousness and perception cannot be explained by mechanical processes: Leibniz proposed a thought experiment in which a brain could be enlarged to the size of a mill, and we would still not be able to find anything to explain, for example, perception. *Leibniz's gap* refers to the problem that mental states cannot be observed by just examining brain processes, which is connected to the hard problem of consciousness in philosophy of mind.[4] The latter invokes the scientific method which we use to explain all the structural and functional properties of the mind, but we still cannot answer why sentient beings have subjective phenomenal experiences. Chalmers [4] states that the easy problems of consciousness explain the following phenomena:

- the ability to discriminate, categorize and react to environmental stimuli,
- the integration of information by a cognitive system,
- the reportability of mental states,
- the ability of a system to access its own internal states,
- the focus of attention,
- the deliberate control of behaviour and
- the difference between wakefulness and sleep.

All of these problems can be explained using computational or neural mechanisms, but the really hard problem of consciousness is the problem of *experience*, i.e. the subjective aspect of it,[5] which still remains an explanatory gap. One's sensation of eating a chocolate bar may be different from another man's, and one can enjoy Stravinsky's *The Rite of Spring* and the other person may hate it. The hard problem of consciousness is the modern version of the centuries-old mind–body problem in philosophy: how to connect our thoughts and consciousness with the brain and the physical body.

Searle's [21] *Chinese Room argument*[6] states that syntax by itself is not sufficient for semantics since a computer can fool a person that it knows Chinese just by following the programmed instructions without knowing it for real, i.e. pure manipulation of symbols may never be true understanding. Unlike the mentioned weak AI system that simulates understanding, the strong-AI position states that AI systems can be used to explain the mind and that the Turing test is adequate to test for the existence of mental states. If we would solve one of **AI**-complete problems, we would have a way to claim we have reached the strong-AI status, or at least crossed a significant

[4]That is, material things like brains, and hence computers, cannot have mental states.

[5]The subjective experiences are usually known in philosophy as *qualia*.

[6]Suppose that we were able to succeed in constructing a computer that seems to understand Chinese. The computer takes Chinese characters as input, follows the programmed instructions and produces other Chinese symbols as an output. Suppose that it does it so competently that it passes the Turing test and convinces a human who speaks Chinese that the program is a human Chinese speaker. Searle then asks the question does the machine really *understand* Chinese, or it is merely simulating that ability.

barrier. Using Yampolskiy's [27] formalization, it has been shown that in that framework any problem solvable by a human oracle could be encoded as an instance of the Turing test, so passing the Turing test seems to be the main step towards achieving the artificial general intelligence. However, according to Searle, the computer may still not truly understand the given task.

11.4 The Walkaround

Shapiro [23] states that solving a problem of one of the main AI-problem areas is equivalent to solving the entire AI problem, i.e. producing a generally intelligent computer program. These areas include natural language, problem-solving and search, knowledge representation and reasoning, learning, vision, and robotics. However, we can see that philosophers and AI researchers managed to pinpoint several key concepts of artificial general intelligence, without the need for statistical calculations of what percentage of AI researchers agrees on what difficult problems are, which is an *informal* part of AI-problem *formalization* in [1]. Generally, solving the **AI**-complete problems using computational methods would certainly fall under the umbrella of weak AI, but it would be still open to philosophical interpretations whether such solutions do constitute *real understanding*, i.e. strong AI.

Strong AI does not have to be superintelligent, only human-like. For example, Trazzi and Yampolskiy [24] introduced *artificial stupidity*, i.e. in order to completely mimic human understanding, supercomputers should not have supercomputer powers, such as the maximum number of operations per second. That means that in order to mimic a human brain, we could pose, for example, that the mentioned $O(n^2)$ method for multiplication needs a comeback since it is a human standard of calculating. Still, as they note, the brain has evolved to achieve some very specific tasks useful for evolution, but nothing guarantees that the complexity of these processes is algorithmically optimal, so the artificial general intelligence could possess a structure that is more optimized for computing than the human brain.

It is interesting to note that humans perform well on some **NP**-complete problems. For example, the travelling-salesman problem which consists in finding the shortest path through a set of points and returning to the initial position[7] was tested on humans, and their solutions were either closed to best-known solutions or were an order of magnitude better than well-known heuristical methods [17]. Even more interesting, an aggregate set of proposed solutions from a group seems to be better than the majority of individual solutions [30], which was tested on a travelling-salesman problem as well. One could posit that combining different machine-learning methods may be close to a human-like solution.

Shahaf and Amir [22] went through a similar path and worked on switching the computational burden between a human and a machine. The complexity of executing an algorithm with a machine M^H is a pair $\langle \phi_H(M^H), \phi_M(M^H) \rangle$, which is a

[7]That is, the decision version tests whether the given route is the shortest route or not.

combination of the complexity of the human part of the work and of the machine's part. For example, optical character recognition is the conversion of printed or handwritten text into machine-encoded text, which is a part of computer-vision issues. Deep-learning methods are usually used for intelligent character or word recognition, where different font styles and different handwriting styles can be learned by a computer during the process. The final complexity of optical character recognition is likely to be $\langle O(1), poly(n)\rangle$. Turing test, which Yampolskiy used as the first step towards reducibility, is reproduced by an n-sentence conversation, which has complexity $\langle O(n), O(n)\rangle$ where the oracle remembers the previous history, $\langle O(n), O(n^2)\rangle$ where the whole conversation history needs to be retransmitted, or $\langle O(n^2), O(n^2)\rangle$ if along with the previous step, a person takes linear time to read the query.

11.5 The Bridge

Let us return to Shapiro's main **AI**-complete areas. First, the goal of natural-language area in AI is to create a program that can use a human language which would be as competent as a human speaker. Unlike natural-language processing, which also encompasses parsing and text-mining methods unrelated to **AI**-complete problems, natural-language understanding deals with reading comprehension, and Yampolskiy considers it an **AI**-hard problem. Methods of solving natural-language-processing problems have been based on shallow models such as support vector machine and logistic regression, trained on high-dimensional and sparse features, but recently deep-learning frameworks based on dense vector representations produce superior results [31]. For example, for text classification (categorizing text into groups) convolutional neural networks, which are very successful in computer vision, were used as well. The main idea is to use feature extraction[8] and classification[9] as a joint task, and to use as many layers as (usefully) possible, along with a hierarchical representation, which can be used to learn the hierarchy of complete sentences [5]. Conneau et al. managed to outperform all previous neural-network models using convolutional neural networks with 29 layers in sentence classification using news and online reviews to extract topics, sentiment analysis and news/ontology categorization.

[8]*Feature extraction* consists of finding the most informative and yet compact set of properties or characteristics for a given problem.

[9]*Classification* is giving a discrete class/category label. Our mapping function needs to be as accurate as possible so that whenever there is a new input data x, we can predict the output variable y, for example, for a picture of a cat, we can put it in a category *cat* and not *dog*. In *supervised machine learning*, where we are training on one (usually larger) dataset and then checking our performance on another dataset, there is also *regression*, where the output variable is numerical or continuous, for example, "the price of this bike is $1500".

Convolutional neural networks[10] have been a bragging point of computer-vision world, so let us examine the state of solving such problems using deep-learning methods. Convolutional neural networks and deep-learning techniques have been successfully used to solve problems in computer vision, especially regarding object recognition, which deals with finding and identifying different objects in images or videos. The usual issues include background noise, occlusions, translations and rotations, but using deep-learning methods these objects still can be recognized. Region-based convolutional neural networks were used by Gu et al. [10] by bridging the gap between image classification and object detection. They focused on localizing objects using a deep network and training the model with only a small quantity of annotated data. The first issue was solved by using the recognition-using-regions methods, in which regions are described by a rich set of cues such as shapes, colours and textures inside them, and then different region weights are learned. Such a solution presents a unified technique for object detection, segmentation and classification.

However, such methods need to be further optimized for more difficult problems. Girschick [9] addressed the deep-learning limitations:

1. training is a multistage pipeline,
2. training is expensive in space and time and
3. object detection is slow.

Even though we are solving computer-vision problems, we are still left with the shackles of computational complexity, both because features that are extracted require hundreds of gigabytes of storage, and because the process may take, for example, 2.5 GPU days for 5000 images and detection takes around a minute for an image using a GPU [15].

Thus, one of the main issues of neural networks is training time. Judd [13] had posited the following question in 1988: *given a general neural network and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the training examples?* Judd has also shown that a problem remains **NP**-hard even if the network needs to produce the correct output for just *two-thirds* of the training examples. Five years later, Blum and Rivest [3] gave us even worse news: for a 2-layer (3-node n-input) neural network, finding the weights that are the best fit for the training set is **NP**-hard, and it is **NP**-complete to decide whether there are weights and thresholds for the three nodes of the network to produce a correct output learned from a trained set. So, even a simple network is **NP**-hard to train. However, that does not mean we cannot achieve satisfactory results. The usual methods [16] that enable successful training are changing the activation function (which defines the output of the node given the set of inputs), over-specification (it seems to be easier to train larger than needed networks) and

[10]Multilayer networks. For example, in computer vision, in face detection, the first layer in a neural network may find regions or edges, the second may find eyes, nose and mouth, the third will make a face contour, etc.

regularization (regularizing the weights so we reduce overfitting[11]) so deep-learning methods are still used to achieve decent results.

So, since the optimization problem for general neural networks is **NP**-hard, the optimization of such a neural network to produce a solution in polynomial time may still seem too far. Contrary to this sceptical news, can we really use deep-learning methods to solve **NP**-complete and **AI**-complete problems efficiently? As a matter of fact yes, for example, gradient-descent[12] methods can provide us with local minima that are good enough, the same way people can solve **NP**-complete problems on a reasonable scale with a satisfactory solution, not optimal. And optimization is just one of the possible things we can do—we can add more machines (thus, more memory = more space) and use better hardware such as GPUs. Why are we requiring that AI does better than we can while at the same time taking human capacities as the epitome of intelligence?

11.6 Multiplying the Multiplication

Milan et al. [19] have shown that using recurrent neural networks by learning from approximate examples produces highly accurate results with minimal computational costs. RNNs are a class of artificial neural networks that learn from prior inputs while producing outputs, while in traditional neural networks we assume that inputs and outputs are independent of each other. But that is not an optimal solution for many natural-language-understanding or image-processing problems, since, for example, if we want to train a model to predict the next word or a phrase in a sentence, we would gain a lot from data regarding previous words. So, recurrent means that for every element the same task is performed and the output depends on previous computations; the same way an **AI**-complete problem can be reduced to a Turing test that has a memory of all previous conversations, which influences the next answer to a tester's question.

Deep-learning and machine-learning methods share one common property, which may also be a common limitation. We want to define how close is our prediction to a correct solution using a loss function and our goal is to minimize the loss. The change of strategy in [19] was not to focus on minimizing the goal function since it may not reflect the network's performance at all in some problems, but to use a problem-specific objective. For example, while solving the travelling-salesman problem, it is not guaranteed that a path that is more similar to the shortest path provides us with a shorter length, since we can replace two short edges in an optimal

[11] Overfitting is when a model corresponds too closely to a particular dataset, which usually means it will fail on more general examples since it contains too many specific parameters. For example, if we were to train a model that can recognize animal and human faces, using pictures of cats, which we described thoroughly to form our relevant attributes, our model could look for pointy ears as a relevant property, and work on cats but not on other animals nor humans (maybe it would work on Vulcans and Elves).

[12] *Gradient descent* is an optimization algorithm for finding the minimum of a function.

travelling-salesman path, and come up with a solution that has a small loss, but also gives us a non-optimal path. So, a problem-specific objective in a supervised manner is computed at each iteration of gradient descent for the approximate solution and our prediction, and in this method the gradient is propagated only if the proposed solution gives a better objective than our predicted solution.

Weston et al. [26] attacked the **AI**-complete problem of question answering using long short-term memory recurrent neural networks (specialized for sequential data) and memory networks (performing matching and inference over previous memories), and have shown that memory networks outperform the other methods, especially taking into account that they perform well at the question answering. They did achieve accuracy over 95% for most of the problems, but they still failed at a number of tasks, and some of these failures were expected due to the insufficient modelling power, such as they perform only two operations maximally, so they cannot handle questions with more than two supporting facts.

The last two scenarios show that problem-specific models outperform general solutions, the same way that some neural networks are better for some problems. It was also demonstrated that people seem to provide better solutions to **NP**-complete problems when viewed as a group, and the common averaged solution outperforms the individual ones. A similar experiment is Google's PathNet, whose task is to discover which parts of the network to reuse for new tasks while learning the user-defined task as efficiently as possible. Fernando et al. [7] claim that for artificial general intelligence it would be efficient if multiple users trained the same giant neural network, without catastrophic forgetting, and with parameter reuse.

Unlike **NP**-complete problems, **AI**-complete problems are not mathematically defined yet although we have mentioned some formalizations. However, using deep-learning methods specialized for different types of problems, modern computing methods, especially using deep learning, are producing highly accurate results and sometimes failing because of specific model restrictions, which seems to mimic human performance as well.

For example, in 2003 Ahn et al. [1] developed CAPTCHA, ***c**ompletely **a**utomated **p**ublic **T**uring test to tell **c**omputers and **h**umans **a**part*, where a user types out the letters of a distorted image. Ye et al. [29] used generative adversarial networks, which are useful when we do not have large training datasets, so the GAN produces lookalike data. Their method solved CAPTCHAs with a 100% accuracy on a number of sites and the algorithm can solve a CAPTCHA within 0.05 of a second on a regular PC. In 2003, Ahn et al. [1] have stated that any program with a high success over a CAPTCHA can be used to solve an unsolved AI problem, so deep-learning methods seem to be on the right track. Either we are still far away from finding a general deep-learning solution for all the problems, or finding a specific solution for distinct problems may be a part of a general solution as well. Learning how to tweak neural networks, how to train them effectively, and which type of propagation to use is, after all, the most human way of solving problems, which may be transferred to self-learning and self-correcting deep-learning methods as well.

Hard **AI**-problems are often similar to **NP**-hard problems, and often **NP**-hard problems coincide with some of the sub-problems of artificial general intelligence.

However, we still do not know if there is an optimal way of solving such problems, but we do know that people and computers, especially using machine-learning and deep-learning methods, can produce sufficiently accurate results for **NP**-complete problems, and that groups of people have greater accuracy than single agents. AI problems seem to be easy for humans, but difficult for machines, and deep-learning methods have shown that neural networks are producing highly accurate results for natural-language understanding and computer-vision problems. For instance, CAPTCHA-type tests relied on computer inability to produce pattern-recognition tasks as accurate as humans can, but recent development shows that deep-learning models can perform with a 100% accuracy. However, due to human error, this does not have to be the case for humans too. The same way that humans differ in their mental abilities, different machine-learning methods differ in their ability to solve a certain problem as well. It seems that the search for universal intelligence is already hard to define for a human level of understanding, let alone for a computer level, which brings us back to the need of inspecting our definitions and formalizations with a *philosophical eye*.

11.7 Eliminating the Human Factor

AI-complete-method solvers are still far away from measuring or detecting internal states, since, for example, feeling pain and knowing about pain are not the same internal states [28]. In Jackson's article [12], the knowledge argument is used to argue against physicalism, which reduces mental phenomena to physical properties. Jackson provides a thought experiment in which a neurophysiologist Mary investigates the world from a black and white room using a black and white monitor and she learns everything about colours and vision, but if she is released from her room, our intuition goes towards the fact that she will actually learn something new, and that all the physical properties are not enough to explain the experience of colour. Computers are getting better at **AI**-complete problems and in **NP**-complete problems as well, but that notion of experience (like to actually *see* the colour in the previous example) is still miles away from being tackled. Yampolskiy [28] postulates that a new category should be devoted to problems of reproducing internals states of a human mind artificially, and he calls that group of problems consciousness-complete or **C**-complete. Such a human oracle would take input as **AI**-complete human oracle, but would not produce any output besides the novel internal state of the oracle. SAT had been shown to be the first **NP**-complete problem and Yampolskiy [27] has conjectured that the Turing test is the first **AI**-complete problem, so he suspects that consciousness will be shown to be the first **C**-complete problem.

Hence, deep-learning methods are constantly improving in different sub-areas of **AI**-complete problems. An example of CAPTCHA has shown that we do not need a human factor at all to solve an **AI**-hard problem, just by using deep-learning methods and in optimal computational complexities, since the amount of data was low enough for exponential-rise issues, but high enough for everyday practical purposes. Recently, researchers [25] used machine-learning methods to train a model on abstracts of scientific (material science) papers. Using word associations, the program was able to predict thermoelectric candidates even though it had not learned the definition of a thermoelectric material. Word2vec[13] was used to analyse relationship between words that were acquired while parsing over three million abstracts. The model was also tested on historical papers and it managed to predict scientific discoveries before they had happened. Therefore, even though we had not achieved artificial general intelligence, it may seem that computers in some areas that include a form of *understanding* do perform as well as we do, and may be able to make discoveries that humans had missed.

Context awareness, unexpected scenarios, Bongard problems[14] and similar issues are still **AI**-hard problems that are getting lots of attention. The only problem that had effectively seen zero progress is even greater than **AI**-complete problems and that seems to be the old philosophical hard problem of consciousness. Dennett [6] states that if Mary from our knowledge argument is really omniscient regarding colour vision, then she already knows how her brain will react and predict the feelings when seeing coloured flowers, having seen neural correlates in other people's brains. So, perhaps, if we would be able to train the network on a large enough amount of data, the notion of *experience* would be instantly reachable. Hence, even though Blum and Rivest [3] have shown that training a 3-node neural network is **NP**-complete, too little attention has been directed towards computational complexity, while defining AI and general-AI issues, and it seems to be the only limiting factor towards achieving *artificial general intelligence*, a machine that has the capacity to understand or learn any intellectual task a human can. Or, as we have seen in [25], maybe even better.

References

1. Ahn LV, Blum M, Hopper N, Langford J (2003) Using hard AI problems for security. In: EUROCRYPT, CAPTCHA
2. Arora S, Barak B (2009) Computational complexity: a modern approach. Cambridge University Press, Cambridge
3. Blum A, Rivest R (1992) Training a 3-node neural network is NP-complete. Neural Netw 5(1):117–127
4. Chalmers D (1995) Facing up to the problem of consciousness. J Conscious Stud 2(3):200–219
5. Conneau A, Schwenk H, LeCun Y (2017) Very deep convolutional networks for text classification. In: Proceedings of the 15th Conference of the European chapter of the Association for computational linguistics: vol I, Long papers. Association for Computational Linguistics, Valencia, Spain, pp 1107–1116
6. Dennett D (1991) Consciousness explained. Little, Brown and Co., Boston
7. Fernando C et al (2017) Pathnet: evolution channels gradient descent in super neural networks. arXiv:1701.08734

[13]These are two-layer neural networks that are trained to reconstruct the context. If you remove a word, it can predict what the words next to it could be, and finally, as a result, words that share common contexts are close together in the vector space.

[14]Two diagrams, where one has a common attribute that is lacking in the other, see [8].

8. Foundalis H, Phaeco: a cognitive architecture inspired by Bongard's problems. PhD thesis
9. Girshick R (2015) Fast R-CNN. In: Proceedings of the 2015 IEEE International conference on computer vision (ICCV), ICCV '15. IEEE Computer Society, Washington, DC, USA, pp 1440–1448
10. Gu C et al (2009) Recognition using regions. In: 2009 IEEE Conference on computer vision and pattern recognition
11. Harvey D, van der Hoeven J (2019) Integer multiplication in time O(n log n). hal-02070778, https://hal.archives-ouvertes.fr/hal-02070778
12. Jackson F (1982) Epiphenomenal qualia. Philos Q 32:127–136
13. Judd S (1988) Learning in neural networks. In: Proceedings of the First annual workshop on computational learning theory, COLT '88. Morgan Kaufmann Publishers Inc, Cambridge, MA, USA, pp 2–8
14. Karatsuba AA (1995) The complexity of computations. Proc Steklov Inst Math 211:169–183
15. Khan S et al (2018) A guide to convolutional neural networks for computer vision. Morgan & Claypool
16. Livni R, Shalev Shwartz S, Shamir O (2014) On the computational efficiency of training neural networks. In: Proceedings of the 27th International conference on neural information processing systems - vol 1, NIPS '14. MIT Press, Cambridge, MA, USA, pp 855–863
17. MacGregor J, Ormerod T (1996) Human performance on the traveling salesman problem. Percept Psychophys 58(4):527–539
18. Mallery JC (1988) Thinking about foreign policy: finding an appropriate role for artificially intelligent computers. Paper presented on the 1988 annual meeting of the International Studies Association
19. Milan A, Rezatofighi SH, Garg R, Dick A, Reid I (2017) Learning in neural networks. In: Proceedings of the First annual workshop on computational learning theory, AAAI '17. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, pp 1453–1459
20. Schönhage A, Strassen V (1971) Schnelle Multiplikation großer Zahlen. Computing 7:281–292
21. Searle J (1980) Minds, brains and programs. Behav Brain Sci 3(3):417–457
22. Shahaf D, Amir E (2007) Towards a theory of AI completeness. In: Commonsense 2007, 8th International symposium on logical formalizations of commonsense reasoning
23. Shapiro SC (ed) (1992) Artificial intelligence. In: Encyclopedia of artifical intelligence, 2nd edn. Wiley, New York, pp 54–57
24. Trazzi M, Yampolskiy R (2018) Building safer AGI by introducing artificial stupidity. arXiv:1808.03644
25. Tshitoyan V et al (2019) Unsupervised word embeddings capture latent knowledge from materials science literature. Nature 571:7
26. Weston J et al (2015) Towards AI-complete question answering: a set of prerequisite toy tasks. arXiv:1502.05698
27. Yampolskiy R, AI-complete, AI-hard, or AI-easy: classification of problems in artificial intelligence. In: The 23rd Midwest artificial intelligence and cognitive science conference, Cincinnati, OH, USA
28. Yampolskiy R, Turing test as a defining feature of AI-completeness. In: Yang X-S (ed) Artificial intelligence, evolutionary computing and metaheuristics
29. Ye G et al (2018) Yet another text captcha solver: a generative adversarial network based approach. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, CCS '18. ACM, New York, NY, USA, pp 332–348
30. Yi SKM, Steyvers M, Lee M, Dry M (2012) The wisdom of the crowd in combinatorial problems. Cogn Sci 36:452–470
31. Young T, Hazarika D, Poria S, Cambria E (2018) Recent trends in deep learning based natural language processing. IEEE Comput Intell Mag 13(3):55–75